Exception Handling in Python

Python

Exceptions

Exception Handling

Try and Except

Nested try Block

Handling Multiple Exceptions in single Except Block

Raising Exception

Finally Block

User Defined Exceptions

Exception

- When writing a program, we, more often than not, will encounter errors.
- Error caused by not following the proper structure (syntax) of the language is called syntax error or parsing error
- Errors can also occur at runtime and these are called exceptions.
- They occur, for example, when a file we try to open does not exist (FileNotFoundError), dividing a number by zero (ZeroDivisionError)
- Whenever these type of runtime error occur, Python creates an exception object. If not handled properly, it prints a traceback to that error along with some details about why that error occurred.

Python 3.6.3 Shell

File Edit Shell Debug Options Window Help

```
Python 3.6.3 (v3.6.3:2c5fed8, Oct  3 2017, 17:26:49) [MSC v.1900 32 bit (Intel)] on win32
Type "copyright", "credits" or "license()" for more information.
>>> a = 5
>>> b = 0
>>> res = a / b
Traceback (most recent call last):
  File "<pyshell#2>", line 1, in <module>
    res = a / b
ZeroDivisionError: division by zero
>>>
```

Exception Handling

- To handle exceptions, and to call code when an exception occurs, we can use a **try/except** statement.
- The **try** block contains code that might throw an exception.
- If that exception occurs, the code in the **try** block stops being executed, and the code in the **except** block is executed.
- If no error occurs, the code in the **except** block doesn't execute.

exception_divide.py - D:/sikander/python/exception_divide.py (3.6.3)

File Edit Format Run Options Window Help

```
#Program to perfrom division and understand exception handling

a = int ( input('Enter the first number ' ) )
b = int ( input('Enter the second number ' ) )

try:
    res = a / b
    print('result = ' , res)
except:
    print('Exception Handled ')

print('End of program')
```

Nested Try Block

```
try:
        num = int ( input('Enter the numerator ') )
        den = int ( input('Enter the denominator ' ) )
        try:
                result = num / den;
                print('Result = '  , result)
        except:
                print('Divide by Zero Error')
except:
        print('Invalid Input')

print('End of Program ')
```

exception_divide.py - D:/sikander/python/exception_divide.py (3.6.3)

File Edit Format Run Options Window Help

```
#Program to know the type of exception

import sys


a = int ( input('Enter the first number ' ) )
b = int ( input('Enter the second number ' ) )

try:
    res = a / b
    print('result = ' , res)
except:
    print('Exception Handled ')
    print("Oops!",sys.exc_info()[0],"occured.")

print('End of program')
```

- A **try** statement can have multiple different **except** blocks to handle different exceptions.

user@varsity-OptiPlex-320: ~/sikander/python/exception

```
import sys
try :
        num = int ( input('Enter the numerator ') )
        den = int ( input('Enter the denominator ' ) )
        result = num / den;
        print('Result = ' , result)
except ValueError:
        print('Invalid Input ')
except ZeroDivisionError:
        print('Divide by Zero Error')

print('End of Program ')
```

~

13,0-1 All

exception_multiple_except2.py - D:/sikander/python/exception_multiple_except2.py (3.6.3)

File Edit Format Run Options Window Help

```
#Program to demonstrate multiple except(catch) blocks

try:

    a = input('Enter the first number ' )
    b = input('Enter the second number ' )
    a = int(a)
    res = a + b
    print('result = ' , res)
except ValueError:
    print('Invalid Input Error')
except TypeError:
    print('Type Error')
except ZeroDivisionError:
    print('Divide by Zero Error')


print('End of program')
```

- Multiple exceptions can also be put into a single **except** block using parentheses, to have the **except** block handle all of them.

```
#Program to demonstrate handling multiple exception types in a single block

import sys

try:

    a = input('Enter the first number ' )
    b = input('Enter the second number ' )
    a = int(a)

    sum = a + b
    print('sum = ' , sum)
    quotient = a // b
    print('quotient = ' , quotient)

except (ValueError,TypeError):
    print('Invalid Input Error',sys.exc_info()[0])

except ZeroDivisionError:
    print('Divide by Zero Error')


print('End of program')
```

```
def divide(num,den):
        res =  num / den
        return res


try :
        num = int ( input('Enter the numerator ') )
        den = int ( input('Enter the denominator ' ) )

        result = divide(num , den)
        print('Result = ' , result)

except ValueError:
        print('Main Block : Invalid Input ')
except ZeroDivisionError:
        print('Main Block : Divide by Zero error')

print('End of Program ')
~
~
```

Raising Exceptions

exception_raise.py - D:/sikander/python/exception_raise.py (3.6.3)

File Edit Format Run Options Window Help

```
try:
    print('Enter the marks ' )
    marks = int( input() )

    if marks < 0 or marks > 100:
        raise ValueError

    #write code to calculate grade

except ValueError:
    print('Input out of range')
```

user@varsity-OptiPlex-320: ~/sikander/python/exception

```
def divide(num,den):
        try:
                res =  num / den
                return res
        except ZeroDivisionError:
                print('Divide Function : Divide by Zero Error')
                return 0

try :
        num = int ( input('Enter the numerator ') )
        den = int ( input('Enter the denominator ' ) )

        result = divide(num , den)
        print('Result = ' , result)

except ValueError:
        print('Main Block : Invalid Input ')
except ZeroDivisionError:
        print('Main Block : Divide by Zero error')

print('End of Program ')
~
```

Raising Exception from Except Block

user@varsity-OptiPlex-320: ~/sikander/python/exception

```
def divide(num,den):
        try:
                res =  num / den
                return res
        except ZeroDivisionError:
                print('Divide Function : Divide by Zero Error')
                raise


try :
        num = int ( input('Enter the numerator ') )
        den = int ( input('Enter the denominator ' ) )

        result = divide(num , den)
        print('Result = ' , result)

except ValueError:
        print('Main Block : Invalid Input ')
except ZeroDivisionError:
        print('Main Block : Divide by Zero error')

print('End of Program ')
```

finally

- To ensure some code runs no matter what errors occur, you can use a **finally** statement.
- The **finally** statement is placed at the bottom of a **try/except** statement.
- Code within a **finally** statement always runs after execution of the code in the **try**, and possibly in the **except**, blocks.

exception_finally.py - D:/sikander/python/exception_finally.py (3.6.3)

File Edit Format Run Options Window Help

```
#Program to demonstrate finally blocks

try:

    a = int ( input('Enter the first number ' ) )
    b = int( input('Enter the second number ' ) )

    res = a / b
    print('result = ' , res)
except (ValueError,TypeError):
    print('Invalid Input Error')
except ZeroDivisionError:
    print('Divide by Zero Error')
finally:
    print('This code will run no matter what')

print('End of program')
```

```
>>>
============== RESTART: D:/sikander/python/exception_finally.py ==============
Enter the first number 4
Enter the second number 2
result =  2.0
This code will run no matter what
End of program
>>>
============== RESTART: D:/sikander/python/exception_finally.py ==============
Enter the first number 4
Enter the second number a
Invalid Input Error
This code will run no matter what
End of program
>>>
============== RESTART: D:/sikander/python/exception_finally.py ==============
Enter the first number 4
Enter the second number 0
Divide by Zero Error
This code will run no matter what
End of program
>>>
```

- Code in a **finally** statement even runs if an uncaught exception occurs in one of the preceding blocks.

exception_finally2.py - D:/sikander/python/exception_finally2.py (3.6.3)

File Edit Format Run Options Window Help

```
try:
    print(1)
    print(10 / 0)
except ZeroDivisionError:
    print(unknown_var)
finally:
    print("This is executed last")
```

Raising Exception

- Raising exception is similar to throwing exception in C++/Java.
- You can raise exceptions by using the **raise** statement

User Defined Exception

```
class InvalidRange(Exception):
        pass


try:
        marks = input('Enter the marks : ')
        marks = int(marks)
        if(marks < 0 or marks > 100):
                raise InvalidRange
        print('Marks = ' , marks)

except ValueError:
        print('Invalid Input')
except InvalidRange:
        print('Input value out of range')
```

~